

Expressões Regulares - Guia de Consulta Rápida

por Aurelio Marinho Jargas e Editora Novatec, ©2001

[Índice](#) [Anterior](#) [Próxima](#)

Pesquisar

Baixar Livro Grátis Em Pdf



iba.com.br

Os Melhores Livros Estão no iba. Compre, Baixe e Leia Agora!

Lista: a exigente [...]


Bem mais exigente que o ponto, a lista não casa com qualquer um. Ela sabe exatamente o que quer, e nada diferente daquilo, *a lista casa com quem ela conhece*.

Ela guarda dentro de si os caracteres permitidos para casar, então algo como `[aeiou]` limita nosso casamento a apenas letras vogais.

No exemplo anterior do ponto, sobre acentuação, tínhamos a ER `n.o`. Além dos casamentos desejados, ela é muito

Está
gostando da
leitura? Está
aprendendo?
Compre o
livro novo que
traz mais
conteúdo, texto atualizado e
ilustrações!





PostgreSQL

Aprenda **PostgreSQL**
com quem faz
na prática!

Instrutores que
atuam no
mercado de TI.

BI
Datawarehouse
Performance
Tuning
PL / SQL
PostGIS
outros ...

CLIQUE AQUI



abrangente, e também casa coisas indesejáveis como neo,
n-o, n5o e n o.

Para que nossa ER fique mais específica, trocamos o ponto pela lista, para casar apenas o não e nao desejados, veja:

```
n[ãa]o
```

E assim como o `n.o`, todos os outros exemplos anteriores do ponto casam muito mais que o desejado, justo pela sua natureza "promíscua".

Por isso que nos exemplos tinha os três pontinhos ... no final?

Exatamente, eles indicam que havia mais possibilidades de casamento. Como o ponto casa com qualquer coisa, ele é nada específico. Então vamos impor limites às ERs:

<code>n[ãa]o</code>	<code>não, nao</code>
<code>[Tt]eclado</code>	<code>Teclado, teclado</code>
<code>e[sx]tendido</code>	<code>estendido, extendido</code>
<code>12[:.]45</code>	<code>12:45, 12.45, 12 45</code>
<code><[BIP]></code>	<code>, <I>, <P></code>

Mas e aquele ponto na ER da hora, não casa com qualquer coisa?

Pegadinha! Não. Registre em algum canto de seu cérebro: *dentro da lista, todo mundo é normal*. Repetindo: *dentro da lista, todo mundo é normal*. Então aquele ponto é simplesmente um ponto normal e não um metacaractere.

No exemplo de marcação `<[BIP]>`, vemos que as ERs são sensíveis a maiúsculas e minúsculas, então se quisermos mais possibilidades, basta incluí-las:

`<[BIPbip]>` ``, `<I>`, `<P>`, ``, `<i>`, `<p>`

Intervalos em listas

Por enquanto, vimos então que a lista abriga todos os caracteres permitidos em uma posição. Como seria uma lista que dissesse que numa determinada posição poderia haver apenas números?

Peraí que essa eu sei... deixa ver... [0123456789]. Acertei?

Sim! Então para casar uma hora, qualquer que ela seja, fica como? Lembre que o formato é `hh:mm`.

Tá. [0123456789][0123456789]:[0123 - Argh! QUE SACO!

Pois é! Assim também pensaram nossos ilustres criadores das ERs, e, para evitar esse tipo de listagem extensa, temos **somente dentro da lista** o conceito de intervalo.

Lembra, que eu disse para você memorizar que dentro da lista, todo mundo é normal? Pois é, aqui temos a primeira exceção à regra. Todo mundo, fora o traço. Se tivermos um traço (-) entre dois

caracteres, isso representa todo o intervalo entre eles.

Não entendeu? É assim, olhe:

`[0123456789]` é igual a `[0-9]`

É simples assim. Aquele tracinho indica um intervalo, então `0-9` se lê: "de zero a nove".

Voltando a nossa ER da hora, poderíamos fazer `[0-9][0-9]:[0-9][0-9]`, mas veja que não é específico o bastante, pois permite uma hora como 99:99, que não existe. Como poderíamos fazer uma ER que case no máximo 23 horas e 59 minutos?

Calma lá... `[012][0-9]:[0-5][0-9]`. Quase, só que tá pegando 29 horas...

Excelente! Com o que aprendemos até agora, esse é o máximo de precisão que conseguimos. Mais adiante, quem poderá nos ajudar será o **ou**. Depois voltamos a esse problema.

Esse intervalo funciona só para números? Algo como `a-z` também existe?

Era isso que eu ia falar agora. Sim, qualquer intervalo é válido, como `a-z`, `A-Z`, `5-9`, `a-f`, `:-@`, etc.

De : até @?

Sim. Por exemplo, se eu quiser uma lista que case apenas letras maiúsculas, minúsculas e números:
`[A-Za-z0-9]`.

Sério, intervalo de : até @?

Sim. Ah! E tem uma pegadinha. Como o traço é especial dentro da lista, como fazer quando você quiser colocar na lista um traço literal?

Sei lá, eu queria saber sobre o intervalo do arroba...

Espere um pouco. Basta colocar o traço no final da lista, assim `[0-9-]` casa números ou um traço. E tem os colchetes, que são os delimitadores da lista. Como incluí-los dentro dela?

O colchete que abre não tem problema, pode colocá-lo em qualquer lugar na lista, pois ela já está aberta mesmo e não se pode ter uma lista dentro da outra.

O colchete que fecha deve ser colocado no começo da lista, ser o **primeiro** item dela, para não confundir com o colchete que termina a lista. Então `[] -` casa um `]` ou um `-`.

Vamos juntar tudo e fazer uma lista que case ambos os colchetes e o traço: `[] [-]`. Calma. Pare, pense, respire fundo, encare esta ER. Vamos lê-la um por um: o primeiro `[` significa que é o começo de uma lista, já dentro da lista, temos um `]` literal, seguido de um `[` literal, seguido de um `-` literal, e por último o `]` que termina a lista. Intuitivo, não? &:)

Tá, confundi tudo, mas que diabos tem entre o : e o @???

Tudo bem, você venceu. Nesse intervalo tem : ; < = > ? @ . Como saber isso? Os intervalos respeitam a ordem numérica da tabela ASCII, então basta tê-la em mãos para ver que um intervalo como `A-z` não pega somente as maiúsculas e minúsculas como era de se esperar.

Para sua comodidade, a tabela está no fim do guia, e nela podemos ver que `A-z` pega também `"[\]^_`"` e não pega os caracteres acentuados como `"áéóõç"`. Infelizmente, não há um intervalo válido para pegarmos todos os caracteres acentuados de uma vez. Mas já já veremos a solução...

Não use o intervalo `A-z`, prefira `A-Za-z`

Dominando caracteres acentuados (POSIX)

Como para nós brasileiros se `a-z` não casar letras acentuadas não serve para muita coisa, temos uns curingas **somente** para usar em listas que são uma mão na roda. Duas até.

Eles são chamados de **classes de caracteres POSIX**. São grupos definidos por tipo, e POSIX é um padrão internacional que define esse tipo de regra, como será sua sintaxe, etc. Falando em sintaxe, aqui estão as classes:

classe POSIX	similar	significa

<code>[:upper:]</code>	<code>[A-Z]</code>	letras maiúsculas
<code>[:lower:]</code>	<code>[a-z]</code>	letras minúsculas
<code>[:alpha:]</code>	<code>[A-Za-z]</code>	maiúsculas/minúsculas
<code>[:alnum:]</code>	<code>[A-Za-z0-9]</code>	letras e números
<code>[:digit:]</code>	<code>[0-9]</code>	números
<code>[:xdigit:]</code>	<code>[0-9A-Fa-f]</code>	números hexadecimais

<code>[[:punct:]]</code>	<code>[.,!?:...]</code>	sinais de pontuação
<code>[[:blank:]]</code>	<code>[\t]</code>	espaço e TAB
<code>[[:space:]]</code>	<code>[\t\n\r\f\v]</code>	caracteres brancos
<code>[[:cntrl:]]</code>	<code>-</code>	caracteres de controle
<code>[[:graph:]]</code>	<code>[^ \t\n\r\f\v]</code>	caracteres imprimíveis
<code>[[:print:]]</code>	<code>[^ \t\n\r\f\v]</code>	imprimíveis e o espaço

Note que os colchetes fazem parte da classe e não são os mesmos colchetes da lista. Para dizer *maiúsculas*, fica `[[:upper:]]`, ou seja um `[[:upper:]]` dentro de uma lista `[]`.

O `[[:upper:]]` é uma classe POSIX dentro de uma lista.

Então, em uma primeira olhada, `[[:upper:]]` é o mesmo que A-Z, letras maiúsculas. Mas a diferença é que essas classes POSIX levam em conta a localidade do sistema.

Atenção para essa diferença, pois a literatura na língua inglesa sempre fala sobre esse assunto muito superficialmente, pois eles não utilizam acentuação e deve ser às vezes até difícil para quem está escrevendo o documento entender isso.

Como nossa situação é inversa, e nossa língua é rica em caracteres acentuados, entender essa diferença é de suma importância.

Como estamos no Brasil, geralmente nossas máquinas estão configuradas como tal, usando números no formato `nnn.nnn,nn`, a data é no formato `dd/mm/aaaa`, medidas de distância são em centímetros e outras coisinhas que são diferentes nos demais países.

Entre outros, também está definido que *áéíóú* são caracteres válidos em nosso alfabeto, bem como

ÁÉÍÓÚ.

Então, toda essa volta foi para dizer que o `[[:upper:]]` leva isso em conta e inclui as letras acentuadas também na lista. O mesmo para o `[[:lower:]]`, o `[[:alpha:]]` e o `[[:alnum:]]`.

Nos Estados Unidos, <code>[[:upper:]]</code> é igual a <code>[A-Z]</code> . No Brasil, <code>[[:upper:]]</code> é igual <code>[A-ZÁÃÂÀÉÊÍÓÕÔÚÇ...]</code>

Por isso para nós essas classes POSIX são importantíssimas, e sempre que você tiver de fazer ERs que procurarão em textos em português, prefira `[[:alpha:]]` em vez de `A-Za-z`, sempre.

Então refazendo a ER que casava maiúsculas, minúsculas e números, temos: `[[:upper:]][[:lower:]][[:digit:]]`, ou melhor: `[[:alpha:]][[:digit:]]`, ou melhor ainda: `[[:alnum:]]`. Todas são equivalentes.

Aaaaaaaah, chega de falar de lista!

Tudo bem, acabou (será?). Mas não se assuste, a lista é o único metacaractere que tem suas próprias regras, funcionando como uma minilinguagem dentro das expressões regulares.

Resumão

- A lista casa com quem ela conhece e tem suas próprias regras.
- Dentro da lista, todo mundo é normal.
- Dentro da lista, traço indica intervalo.
- Um `-` literal deve ser o último item da lista.
- Um `_` literal deve ser o primeiro item da lista.
- Os intervalos respeitam a tabela ASCII (não use `A-z`).

- [:classes POSIX:] incluem acentuação, **A**-**Z** não.



[Índice](#) [Anterior](#) [Próxima](#)

[Errata](#) — [Deixe sua opinião](#) — [Site do autor](#)